AIRES, ZHAireS y RASPASS: Cuentas y plantillas

Sergio Cabana Freire

30 de octubre de 2022

Voy a recoger aquí las cuentas que vaya haciendo sobre la geometría de los programas. También incluiré algunas normas que se me ocurran para nombrar ficheros de manera eficiente, y una breve descripción de los módulos con los que trabajo. Lo que sigue está pensado para AIRES 19.04.08 (22/Oct/2021) con la extensión ZHAireS 1.0.30a (30/Jun/2022), que incluye RASPASS[1]. Las diferencias con versiones anteriores estarán en la geometría (aunque los cálculos podrán adaptarse, simplemente fijando algunos parámetros que en RASPASS son libres) y en alguna tabla o instrucción IDL nueva, nada más.

Referencias importantes:
Página web de AIRES / ZHAireS. Documentación actualizada
Presentación de RASPASS. Matías Tueros, ARENA 2022
Repositorio GitHub (con algunas plantillas y códigos que pueden ser útiles)

Índice

[1] *Raspass is an Aires Special Primary for Atmospheric Skimming Showers*. Primera sigla recursiva observada en la naturaleza.

1. Geometría en RASPASS

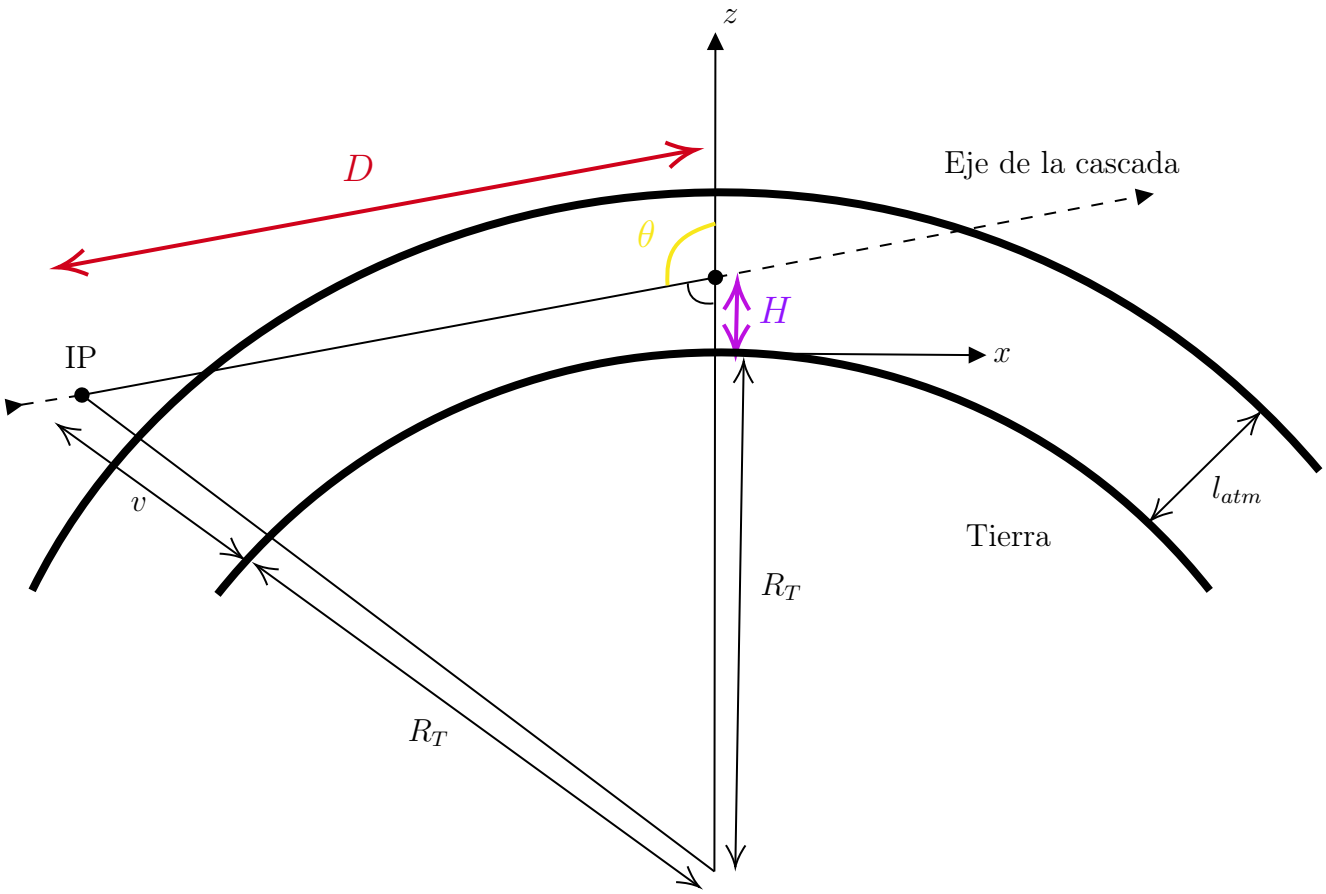

Figura 1.1: Esquema de la geometría en RASPASS

Tenemos tres parámetros para determinar la geometría en RASPASS:

- RASPASSDistance, D. Distancia a lo largo del eje desde el punto de inyección (IP) hasta la intersección con la vertical del observador.
- RASPASSHeight, H. Altura a la que pasa el eje de la cascada sobre la vertical del observador.
- PrimaryZenAngle, θ. Ángulo entre el eje de la cascada y la vertical del observador.

Esta geometría puede reducirse a casos más sencillos:

- Cascadas hacia abajo (normales): $H = 0$, $\theta < 90°$.
- Cascadas hacia arriba: $H = 0$, $D \leq 0$, $\theta > 90°$ (Inyección a nivel del suelo o por encima)

Todo lo que sigue es trigonometría sencilla[2]:

$$(R_T + v)^2 = (R_T + H)^2 + D^2 + 2D(R_T + H)\cos\theta \tag{1.1}$$

[2] $\sin(\pi - x) = \sin x$; $\cos(\pi - x) = -\cos x$

1.1. Altura del punto de inyección

Despejando de (1.1):

$$v = \sqrt{(R_T + H)^2 + D^2 + 2D(R_T + H)\cos\theta} - R_T \tag{1.2}$$

1.2. Valor de D en función de v (y demás parámetros RASPASS)

La ecuación (1.1) es una cuadrática en d:

$$D = -(R_T + H)\cos\theta + \sqrt{(R_T + H)^2\cos^2\theta - [H^2 - v^2 + 2R_T(H - v)]} \tag{1.3}$$

1.3. Altura en la atmósfera, h, en función de distancia recorrida a lo largo del eje desde IP, L

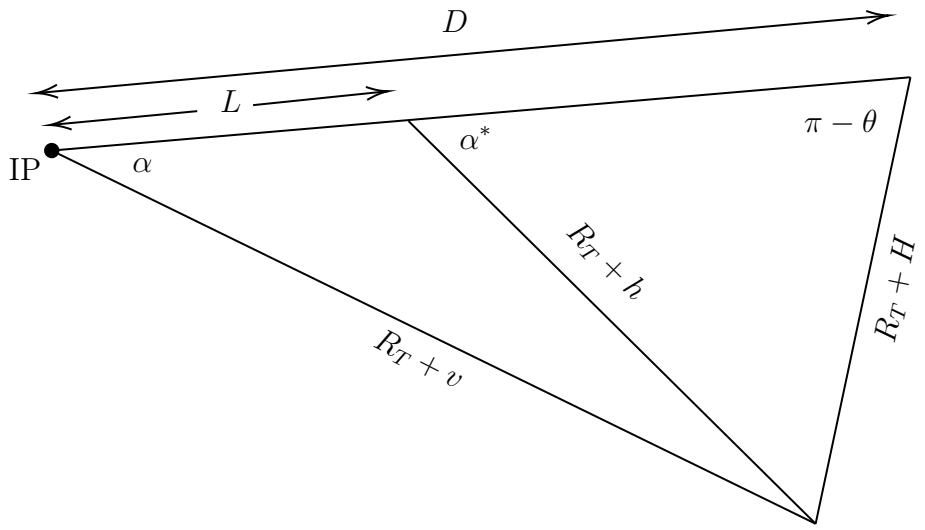

Figura 1.2: Detalle del triángulo en la Fig. 1.1.

Aplicando el teorema del coseno:

$$(R_T + h)^2 = (R_T + v)^2 + L^2 - 2L(R_T + v)\cos\alpha \tag{1.4}$$

$$h = \sqrt{(R_T + v)^2 + L^2 - 2L(R_T + v)\cos\alpha} - R_T \tag{1.5}$$

v puede obtenerse de los inputs de RASPASS (1.2). Falta calcular $\cos\alpha$. Aplicamos el teorema del seno y elevamos al cuadrado:

$$\frac{(R_T + H)^2}{1 - \cos^2\alpha} = \frac{(R_T + v)^2}{\sin^2(\pi - \theta)} = \frac{(R_T + v)^2}{\sin^2\theta} \tag{1.6}$$

$$\cos\alpha = \sqrt{1 - \left(\frac{R_T + H}{R_T + v}\right)^2 \sin^2\theta} \tag{1.7}$$

Las ecuaciones (1.5) y (1.7) forman el resultado. La altura h (1.5) presenta un mínimo cuando:

$$\frac{dh}{dL} = 0 \implies L_{min} = (R_T + v)\cos\alpha \tag{1.8}$$

Es trivial comprobar que el valor mínimo de la altura es:

$$h_{min} = (R_T + v)\sin\alpha - R_T = (R_T + H)\sin\theta - R_T \tag{1.9}$$

La interpretación para los casos en que la expresión anterior devuelve valores negativos es que, simplemente, la trayectoria *cruza* la superficie de la Tierra. En esos casos, el valor físico es $h_{min} = 0$

1.4. Materia atravesada en función de distancia recorrida a lo largo del eje desde IP

La cantidad de materia atravesada es:

$$X(L)\ \left[\mathrm{g/cm^2}\right] = \int_0^L \rho(h(l))dl \tag{1.10}$$

donde dl es el elemento de línea a lo largo del eje. Es fácil ver que, al avanzar a lo largo del eje una distancia dl, la altura cambia en:

$$dh = dl\cos\alpha^*(h) \tag{1.11}$$

donde α^* es el ángulo cenital *local*, i.e., entre una distancia recorrida L y $L+dl$ (Fig. 1.2). Planteando el teorema del seno como en (1.6), es inmediato llegar a que:

$$X(L) = \int_{h(L)}^{v} \rho(h')\frac{dh'}{\sqrt{1 - \left(\frac{R_T+H}{R_T+h'}\right)^2 \sin^2\theta}} \tag{1.12}$$

Debe usarse con cuidado la expresión anterior, que supone que el límite inferior de la integral es menor a v, i.e., *integramos hacia arriba en la atmósfera*. Si tenemos cascadas hacia abajo, en que avanzar a lo largo del eje implica reducir la altura, aparecerá un signo extra. Peor aún, en cascadas estratosféricas la altura primero decrece y luego crece, así que según la región habrá un signo o no.

2. Normas para nombrar *Tasks* en AIRES

Los archivos .inp y outputs de AIRES pueden dar lugar a bastantes dolores de cabeza, a la hora de identificar qué archivos son los correspondientes a qué simulación y viceversa. Se me han ocurrido unas normas para poner nombres a los *Tasks* en AIRES de manera sistemática. La idea es que viendo el nombre de un archivo podamos identificar inmediatamente la simulación en cuestión y sus parámetros claves, además de poder hacer de manera más sencilla gráficas y análisis posteriores (ya

que teniendo unas normas para los nombres de archivos, podremos simplemente escribir un código que haga el trabajo de buscar los ficheros de datos necesarios para la gráfica que queremos, por ejemplo).

El formato típico para un *task* será:

`TrajectoryType_Primary_Energy_TrajectoryParams_ExtraComments`

Es decir, indicaremos:

- Indicador de trayectoria: Si tenemos una cascada normal (*hacia abajo*), escribiremos `dg_` (down-going). En otro caso, si hemos usado una primaria introducida mediante la instrucción adicional `AddSpecialParticle` y un código *externo* a AIRES, lo indicaremos adecuadamente. Los códigos más habituales (en mi caso) serían: `upr_...` (para uprimary), `RAS_...` (para RASPASS)
- Partícula primaria: Siempre con dos caracteres o más.

No válido	`p_...` , `e_...`
Válido	`prot_...` , `elec_...` , `gamma_...` , `iron_...`

  La norma general para las partículas comunes será usar `prot`, `elec`, `gamma`. Para las demás partículas y núcleos, siempre el nombre completo: `muon`, `pion`, `kaon`, `lead`, ...

- Energía de la partícula primaria: En eV con formato exponencial. Por ejemplo, ...`_1e18eV_`..., ...`_5.7e15eV_`...
- Trayectoria: Depende del tipo de partícula, ya que añadir primarios especiales normalmente implica cambiar los parámetros que definen la trayectoria. En cualquier caso, los ángulos se indicarán en grados y las distancias en km (con las unidades explícitas). Para distinguir cenit de azimut, escribiremos las unidades de éste último como `pdeg` (añadimos una `p` de *phi*) Todos los parámetros se indicarán con al menos dos cifras numéricas y se separarán con `_`. En los casos habituales:

	Norma
AIRES (normal)	..._InjectionAltitude_Zenith_Azimuth_...
uprimary	..._ALTITUDE_Zenith_Azimuth_...
RASPASS	..._RASPASSDistance_RASPASSHeight_RASPASSTimeShift_Zenith_Azimuth_...

	Ejemplo
AIRES (normal)	..._100km_65deg_00pdeg_...
uprimary	..._00km_95deg_00pdeg_...
RASPASS	...1500km_35km_00ns_93.5deg_00pdeg_...

  Las cascadas hacia arriba tendrán un ángulo cenital $> 90°$ por convenio. Por defecto, suponemos que el azimut es magnético (si fuera geográfico, añadimos una `G` antes de las unidades.)

Como norma general para otros casos, deben indicarse todos los parámetros que describan las trayectorias.

- Comentarios extra: Para incluir cualquier información relevante para identificar la simulación y sus condiciones. Algunos ejemplos pueden ser:

  1. *La simulación incluye cálculo de emisión electromagnética (en tiempo, frecuencias o ambos)*: ...`_ZHS(tfb)`
  2. *Índice de refracción fijado*: ...`nconst`
  3. *Se ha fijado un campo magnético constante*: ...`fixB`
  4. *Se promedian 5 cascadas*: ...`_05show`
  5. *La cascada se simula en Malargüe*: ...`_SiteMalargue`
  6. *Se utiliza el modelo de atmósfera 3*: ...`_Atmos03`
  7. *El suelo está a* 35 m *a.s.l.*: ...`_grd35m`

  Como recomendación, está bien usar más de dos cifras para cualquier magnitud numérica.

PELIGRO: Existe un límite de 64 caracteres para la longitud de los nombres para tasks, si se supera AIRES truncará la cadena de caracteres y perderemos información en los comentarios extra.

Algunos ejemplos pueden ser:

- Cascada iniciada por un protón de 100 TeV, inyectado a 100 km de altura con ángulos $\theta = 65°$, $\varphi = 0°$, con el suelo a 100 m a.s.l.

  `dg_prot_1e17eV_100km_65deg_00pdeg_grd100m`

- Cascada hacia arriba iniciada por un electrón de $1,5$ EeV, interaccionando a $1,$ km del suelo, con ángulos $\theta = 95°$, $\varphi = 65°$ (geográfico), calculando emisión en radio en dominio temporal con índice de refracción constante:

  `upr_elec_1e18eV_01km_95deg_65Gpdeg_ZHSt_nconst`

- Cascada estratosférica iniciada por un fotón (RASPASS) de 1 EeV, a ángulo cenital $93,25°$ y azimut 0°. La trayectoria cruza la vertical del observador (en el Polo Sur) a 35 km de altura (RASPASSHeight) y el primario se inyecta a 1462,5 km de dicho cruce (RASPASSDistance). No se impone TimeShift, se calcula emisión en radio en ambos dominios con un campo magnético fijado.

  `RAS_gamma_1e18eV_1462.5km_35km_00ns_93.25deg_00pdeg_SouthPole_ZHSb_fixB`

  En este caso se pasa de 64 caracteres y se pierden comentarios extra. Nos encontraríamos esto en los outputs:

  `RAS_gamma_1e18eV_1462.5km_35km_00ns_93.25deg_00pdeg_SouthPole_ZH`

  Hay que recortar el nombre eliminando los comentarios menos relevantes (`_SouthPole`, por ejemplo, ya que el campo magnético que afecta a la simulación de emisión en radio está fijado previamente).

3. Simulaciones y análisis de outputs con mazepin

He preparado un módulo en Python (más bien un script con no se cuántas definiciones de funciones) para preparar simulaciones con AIRES, ZHAireS y RASPASS; y poder sacar gráficas rápidamente de los outputs. He dedicido llamarlo *Module for an Aires and Zhaires Environment in PyhtoN*, o para abreviar, `mazepin`. Aquí me voy a limitar a dar una breve descripción de lo que hace cada función y qué es cada argumento. En el repositorio GitHub hay un script con ejemplos de uso de las funciones que pueden resultar más útiles. Para usar el módulo, creo que lo más adecuado es limitarse a copiar, pegar y adaptar los ejemplos más que entender lo que se recoge aquí. Por lo menos es lo que haré yo, si estoy escribiendo esto es para tener una ayuda en el futuro, cuando no sepa qué hace cada cosa.

Disclaimer: Estoy escribiendo esto en la versión 0.8.2 del módulo, mi plan es añadir más funciones para tratar los outputs de ZHAireS

3.1. Funciones para cálculos geométricos

El módulo incluye una serie de funciones sobre la geometría de las cascadas y la atmósfera. La mayoría de ellas están pensadas para trayectorias RASPASS, porque pueden ser aplicadas fácilmente a casos más sencillos de cascadas cuyo eje intercepta el suelo.

- `gcm2toh(t)`

  Convierte profundidad vertical `t` $\equiv X_v$ [g/cm^2] a altura en km en la atmósfera, mediante la parametrización de Linsley.

- `htogcm2(h)`

  La inversa de la función anterior, convierte altura `h` [km] a profundidad vertical X_v [g/cm^2]

- `CerenkovRing(Xmax, ground, theta, UseSlant = True)`

  Devuelve por pantalla los parámetros de la elipse Čerenkov a nivel del suelo (aproximados y exactos), para cascadas hacia abajo.

  - `Xmax`: profundidad del máximo [g/cm^2] medida sobre el eje de la cascada (si `UseSlant = True`) o en la vertical (si `UseSlant = False`).
  - `ground`: altura del suelo sobre el nivel del mar, en m.
  - `theta`: ángulo cenital del eje de la cascada, en grados.

- `cos_localtheta(h, theta, RASPASSHeight = 0.)`

  Devuelve el coseno del ángulo cenital local $\cos\alpha^*$ a una altura `h` [km] para una trayectoria de ángulo cenital `theta` [deg], y RASPASSHeight en km (por defecto, para cascadas que interceptan con el suelo). Véase la Fig. 1.2, ec. (1.11)

- `RAS_Dist(h, theta, RH = 0.)`

Devuelve la RASPASSDistance tal que, para una trayectoria de ángulo cenital `theta` [deg] y RASPASSHeight `RH` [km] (por defecto 0, cascadas interceptando con el suelo), la altura de inyecció en la atmósfera es `h` [km]. Esta función permite convertir trayectorias *normales*, definidas por una altura de inyección y un ángulo cenital, a trayectorias RASPASS.

- `h_IP(RD, RH, theta)`

  Devuelve la altura en la atmósfera [km] del punto de inyección de una trayectoria RASPASS con RASPASSDistance `RD` [km], RASPASSHeight `RH` [km] y ángulo `theta` [deg]. Puede considerarse la inversa de la función anterior.

- `h_RAS(L, RD, RH, theta)`

  Devuelve la altura en la atmósfera del punto a una distancia a lo largo del eje `L` [km] del punto de inyección de una trayectoria RASPASS con RASPASSDistance `RD` [km], RASPASSHeight `RH` [km] y ángulo `theta` [deg]. Básicamente implementa los resultados (1.5), (1.7)

- `dist_to_Xs(dist, RD, RH, theta, step = .025)`

  Conversor de distancia recorrida a lo largo del eje desde inyección `dist` [km] a materia atravesada a lo largo del eje X_s [g/cm^2] para una trayectoria RASPASS con RASPASSDistance `RD` [km], RASPASSHeight `RH` [km] y ángulo `theta` [deg]. Hace la integral (1.12) teniendo en cuenta que la altura puede crecer o decrecer según el tipo de trayectoria.

  La función divide la distancia recorrida en intervalos de longitud `step` (por defecto 25 m) y aproxima la integral por una suma, empleando la densidad asociada a la parametrización de Linsley (definida en el módulo auxiliar `mazepin_aux`)

- `atmos_size(RH, theta, atmos_height = 100, stop = 'atmos')`

  Devuelve la distancia recorrida total y la materia atravesada por una trayectoria RASPASS de RASPASSHeight `RH` [km] y ángulo `theta` [deg].

  La función ajusta RASPASSDistance para que el comienzo de la trayectoria sea el límite superior de la atmósfera, situado en `atmos_height` (por defecto 100 km). Hay tres opciones para el parámetro `stop`:

  - `stop = 'atmos'` (por defecto): devuelve la distancia recorrida y la materia atravesada a lo largo de la trayectoria desde la inyección en el límite superior de la atmósfera hasta la salida de la misma (i.e., hasta que se alcanza `atmos_height` de nuevo.)
  - `stop = 'zaxis'`: devuelve la distancia recorrida y la materia atravesada a lo largo de la trayectoria desde la inyección en el límite superior de la atmósfera hasta el punto de corte con la vertical del observador
  - `stop = 'hmin'`: devuelve la distancia recorrida y la materia atravesada a lo largo de la trayectoria desde la inyección en el límite superior de la atmósfera hasta el punto de mínima altura en la atmósfera, ec. (1.9).

  En el caso de cascadas que interceptan el suelo, el parámetro `stop` no es relevante, se devuelve distancia recorrida y materia atravesada desde inyección al suelo.

- `Xs_to_dist(X, RD, RH, theta, atmos_height = 100)`

  Conversor de materia atravesada a lo largo del eje `X` [g/cm^2] a distancia recorrida desde inyección [km]para una trayectoria RASPASS con RASPASSDistance `RD` [km], RASPASSHeight `RH` [km] y ángulo `theta` [deg].

  Para la trayectoria de interés, la función llama a `atmos_size` para obtener la distancia total que se recorre en la atmósfera y la materia total que se atraviesa. Con esos dos valores, se genera un array de datos de distancia (desde 0 hasta la distancia máxima dentro de la atmósfera) y el array de materia atravesada correspondiente, obtenido con `dist_to_Xs`. Ambos arrays se utilizan para obtener una función interpolante $d(X_s)$, a la que se llama para devolver el output de esta función.

  La interpolación puede presentar muchos problemas en el *principio* y *final* de las trayectorias, i.e., en las zonas de mayor altura, en las que $X_s(d) \sim cte.$ y la interpolación inversa se vuelve muy difícil.

3.2. Funciones para manejar outputs de AIRES

Las tablas de datos que devuelve AIRES tienen el siguiente formato:

```
Task_name.t(extension)
```

donde `Task_name` identifica, más o menos, la simulación que se ha hecho y `extension` es un número de 4 cifras que identifica el tipo de tabla y la partícula correspondiente.

El objetivo de las funciones siguientes es que, cuando tenemos en una carpeta muchos archivos de este tipo, podamos producir gráficas de manera automática sin más que *pedir* la simulación que nos interesa (i.e., describiendo `Task_name`), y las tablas y partículas que queremos.

- `codes()`

  El formato para las extensiones de tablas de AIRES puede dar dolores de cabeza (al menos a mí). Hay 16 tipos de tablas de datos para 23 tipos de partículas, que se cruzan en no se cuántas tablas, cada una identificada con un número de 4 cifras.

  Para simplificar esto, el módulo identifica las tablas con un número del 0 al 16 y las partículas con un número del 0 al 22. Esta función devuelve la lista completa de tablas y partículas disponibles, con su número asociado.

  Nota: En el módulo, hay 17 tipos de tablas (del 0 al 16), una más que en AIRES. Eso es porque he incluido la opción de tener datos de desarrollo longitudinal de la energía por partícula, dividiendo los datos de las tablas de desarrollo de energía entre los de desarrollo del número de partículas.

- `table_finder(tab, part)`

  Permite encontrar la extensión de una tabla concreta identificada por un número `tab` (del 0 al 16) para un tipo de partícula `part` (número del 0 al 22). De ese modo, si queremos encontrar la

tabla de *Desarrollo longitudinal de piones cargados*, en lugar de ir al manual hasta encontrar el código `1211` (o aprendérnoslo de memoria), podemos simplemente llamar a esta función como `table_finder(0, 17)`, donde 0 el el código de *desarrollo longitudinal* y 17 es el código de $\pi^+\pi^-$.

Con una llamada a `codes`, pueden verse todos los códigos, así que me parece una herramienta al menos útil.

- `pathfinder(rootdir, tabs, part, sim = [''], sep = [''], verbose = False)`

  Encuentra las rutas de los archivos que le pidamos. Para *pedir* archivos, tenemos que dar los siguientes inputs:

  - `rootdir`: El directorio donde se encuentran nuestros archivos, sin importar que haya más.
  - `tabs`: Una lista con los códigos de las tablas que queremos. Por ejemplo, si queremos tablas de desarrollo longitudinal, `tabs = [0]`
  - `part`: Una lista con los códigos de las partículas para las que queremos las tablas. Por ejemplo, si queremos tablas de e^+e^- y $\mu^+\mu^-$, `tabs = [15, 16]`
  - `sim`: Una lista con los strings que todos los archivos devueltos deben contener en su nombre, y que por tanto identifican el *TaskName* y por ello la simulación. Por ejemplo, si tenemos una serie de simulaciones RASPASS de protones de 1 EeV, inyectados con RASPASSDistance 1500 km y ángulo $\theta = 93°$ (y usamos las normas de la sección 2), pediríamos:
    `sim = ['RAS', 'prot', '1e18eV', '1500km', '93deg']`
    Por defecto `sim= ['']`, así que todos los archivos con la extensión de las tablas pedidas saldrían como output.
  - `sep`: Una lista con los strings que pueden cambiar en los nombres de los ficheros, y que podemos utilizar para graficar resultados en los que varía un parámetro. Por ejemplo, si en el caso anterior hemos hecho simulaciones para distintas RASPASSHeight, podemos separar los archivos pidiendo:
    `sep = ['20km', '30km']`
    Por defecto `sep = ['']`, así que no se guarda ninguna información de este estilo.

  La llamada a esta función devuelve una lista con elementos del tipo:

  ```
  [tab, part, sep, path]
  ```

  Como ejemplo, si en una carpeta `Outputs/` tenemos, entre muchos, los ficheros siguientes:

  ```
  RAS_prot_1e18eV_1500km_20km_00ns_93deg_00pdeg.t1205  # e+e-

  RAS_prot_1e18eV_1500km_30km_00ns_93deg_00pdeg.t1207  # mu+mu-

  RAS_prot_1e18eV_1500km_20km_00ns_93deg_00pdeg.t1205  # e+e-

  RAS_prot_1e18eV_1500km_30km_00ns_93deg_00pdeg.t1207  # mu+mu-
  ```

y llamamos a `pathfinder` con los argumentos que se comentaron, obtendríamos:

```
[['0', '15', '20km', 'Outputs/RAS_prot_1e18eV_1500km_20km_00ns_93deg_00pdeg.t1205'],
['0', '15', '30km', 'Outputs/RAS_prot_1e18eV_1500km_30km_00ns_93deg_00pdeg.t1205'],
['0', '16', '20km', 'Outputs/RAS_prot_1e18eV_1500km_20km_00ns_93deg_00pdeg.t1207'],
['0', '16', '30km', 'Outputs/RAS_prot_1e18eV_1500km_30km_00ns_93deg_00pdeg.t1207']]
```

- `readfile(file)`

  Lee un archivo de datos de AIRES con ruta `filename`, y devuelve la matriz de datos (sin la primera columna, que es el número del bin del histograma) y, si encuentra el valor entre los comentarios, la profundidad *slanted* del nivel del suelo. En caso de no hacerlo, devuelve un `'None'`

- `Aires_data(data, error_type = 'sigma', UG = False, slant = False, RASPASS = False, Distance = False, first_int = True, traject = [])`

  Esta función prepara los datos de un archivo de output de AIRES para graficarlos, y devuelve además un *label* adecuado para los datos y ejes del gráfico.

  Los inputs son:

  - `data`: Es una lista de la forma `[tab, part, sim, sep]`, i.e., cada uno de los elementos de un output de `pathfinder`
  - `error_type`: el tipo de error que queremos graficar (en el caso de que el fichero de datos lo contenga, como en promedios de cascadas). Las opciones son `'sigma'` (para desviación estándar, por defecto) y `'RMS'`.
  - `UG`: bool con el que indicamos si los datos son de una cascada *upgoing*.
  - `slant`: bool con el que indicamos si los datos contienen profundidades medidas a lo largo del eje de la cascada.
  - `RASPASS`: bool con el que indicamos si los datos son de una simulación RASPASS.
  - `Distance`: bool con el que indicamos si queremos convertir los valores de profundidad a distancia recorrida. Para cascadas que interceptan con el suelo, debe usarse con `slant = True` (ya que se emplea `Xs_to_dist`), mientras que en RASPASS debe usarse con `slant = False`, ya que la opción por defecto para exportar las tablas ya devuelve distancia en el eje x (i.e., sin el `Opt a` en el input file de AIRES).
  - `first_int`: bool con el que indicamos que queremos comenzar a medir distancias o profundidades atravesadas desde el punto de primera interacción, y no desde el límite superior de la atmósfera (*downgoing*) o el suelo (*upgoing*). Este parámetro no se usa si `RASPASS = True`.
  - `traject`: lista con los parámetros que describen la trayectoria. Para cascadas *downgoing* o *upgoing*, debemos dar `[injection_height(km), theta(0-90deg)]`, mientras que para RASPASS debemos indicar `[RASPASSDistance(km), RASPASSHeight(km), theta(deg)]`

  La función se encargará de hacer los cálculos y conversiones oportunas, y buscará *labels* adecuados para el set de datos, y los ejes del gráfico. El output es de la forma:

`[x_data, y_data, yerr_data, label], x_axis_label, y_axis_label`

- `Aires_plot(input_data, error_type = 'sigma', UG = False, slant = False, RASPASS = False, Distance = False, first_int = True, trajects = [], xlim = [], ylim = [], xscale = 'linear', yscale = 'linear', autolabel = True, graph_type = 'step', labels = [], size = 12, legend = True, title = '', loc_leg = 'best', fmt = '-', marker = 'o', linewidth = 2.)`

  Función para graficar outputs de AIRES de manera automática. Los inputs son:

  - `input_data`: El output directo de `pathfinder`, que habremos conseguido identificando la simulación, las tablas que queremos y los parámetros que varían, de haberlos.
  - `error_type, UG, slant, RASPASS, Distance, first_int, trajects`: Los mismos inputs que en `Aires_data`, con la diferencia de que ahora `trajects` es una lista, cuyos elementos son el `traject` *simple* de `Aires_data` asociado a cada archivo que se va a graficar (es un poco trabalenguas, pero tengo una función que saca esto por si acaso). Esta función coge los archivos desde `input_data` y los pasa a través de `Aires_data` con las condiciones indicadas y el `traject` correspondiente, dado en `trajects`.
  - `autolabel`: bool con el que indicamos si queremos que la función ponga leyendas automáticamente. En el caso de indicar un `False`, debemos introducir como input los labels (en el orden de `input_data`) que queremos, en la lista `labels`.
  - El resto de inputs son parámetros clásicos de `matplotlib.pyplot`. Los únicos que quizá vale la pena comentar son `size` (tamaño de fuente) y `graph_type` (tipos de gráfica, sólo están implementados el histograma `'step'` y el plot con puntos y barras de error `'errorbar'`)

  Esta función se encarga de todo el trabajo de graficar outputs, con sólo identificar qué simulación y tablas nos interesan:

  $$\left.\begin{matrix}\text{Simulación}\\ \text{Tablas}\end{matrix}\right\} \longrightarrow \texttt{pathfinder} \longrightarrow \underbrace{\texttt{Aires\_plot} \longrightarrow}_{\texttt{Aires\_data}} \text{Gráfica}$$

- `traject_finder(files, RASPASS = False, UG = False)`

  Dando como input (`files`) la salida de `pathfinder`, devuelve la lista de trayectorias `trajects` que necesita `Aires_plot`. `RASPASS` y `UG` (upgoing) son bools que indican el tipo de trayectoria. Esta función sólo funciona con el criterio para nombrar *tasks* de la sección 2.

Aquí incluiré las descripciones de funciones para trabajar con outputs de ZHAireS, cuando las prepare

3.3. Funciones para preparar y correr simulaciones

He incluido una serie de funciones capaces de crear input files de AIRES y correrlos en remoto. Aunque las definiciones son un poco brutas, creo que pueden ser útiles a la hora de preparar trabajos *pesados*.

- `input_file(task_name, basic, trajectory, sim_control, RASPASS = False, upgoing = False, ZHAireS = False, ZHAireS_control = [], exports = [], extra = [], save_path = '')`

  Esta función permite crear input files de AIRES, con la opción de incluir cálculos de ZHAireS, y las partículas especiales que se consiguen con los ejecutables `uprimary` (cascadas hacia arriba) y `RASPASSprimary` (trayectorias RASPASS). Los inputs son:

  - `task_name`: Nombre del *task* (sin extensión `.inp`), recomendable seguir las normas de la sección 2.
  - `basic`: Una lista de strings que controla los parámetros básicos de la simulación. Sus elementos son:

    `[Primary, Energy (eV), GeomagneticField, PropagatePrimary, Site, Ground]`

    Por ejemplo, si queremos un primario protón de 100 PeV, con el campo magnético *encendido*, sin propagar el primario, en el Polo Sur con el suelo a 0 m de altura sobre el nivel del mar:

    `['Proton', '1e18', 'On', 'Off', 'SouthPole', '0 m']`

    Si queremos dejar algo por defecto, simplemente ponemos un string vacío. Todas las opciones de AIRES pueden escribirse aquí (o al menos deberían, la función está pensada para casos simples). El argumento `Ground` debe indicarse con unidades, ya sean m ó g/cm^2. Por defecto, se escoge la fecha 11/07/2010.

    `['Proton', '1e18', '50 uT 0 deg 0 deg', 'On', '', '']`

  - `trajectory`: Lista con los parámetros de la trayectoria. Según tengamos cascadas hacia abajo, hacia arriba o RASPASS, indicaremos respectivamente:

    ```
    [injection_height(km), theta(deg), phi(deg)]
    [ALTITUDE(km), theta(deg), phi(deg)]
    [RASPASSDist(m), RASPASSHeight(m), RASPASSTimeShift(ns), theta(deg), phi(deg)]
    ```

    En el último caso, **todo con 2 cifras decimales**.
  - `sim_control`: Parámetros que controlan la simulación:

`[TotalShowers, RunsPerProcess, ShowersPerRun, RandomSeed, ObservingLevels, ThinningEnergy, ThinningWFactor, SaveInFile grdpcles, SaveInFile lgtpcles, ElectronCutEnergy, ElectronRoughCut, GammaCutEnergy, GammaRoughCut]`

Son muchas opciones, pero en el módulo auxiliar `mazepin_aux` se incluyen definiciones de opciones estándar `default_sim_control` y `default_sim_control_ZHS` (adaptado para simulaciones con ZHAireS, thinning más grande)

- `RASPASS`, `upgoing`, `ZHAireS`: bools para controlar si la partícula primaria es especial (e incluir los ejecutables correspondientes), y para incluir simulaciones de ZHAireS.
- `ZHAireS_control`: En el caso `ZHAireS = True`, este objeto controla los cálculos con ZHAireS. Tiene la forma:

  `[FresnelTime, FresnelFreq, list_of_antenna_xyz, list_of_freq, extra_directs]`

  Por ejemplo, si sólo queremos cálculo en frecuencias, en dos antenas situadas en $(0, 0, 0)$ y $(10, 20, 30)$, a 100 y 300 MHz, y usar la directiva `RemoveZeroOutput`; tendremos:

  `['Off', 'On', [[0,0,0],[10,20,30]], [100,300], ['RemoveZeroOutput On']]`

- `exports`: Lista con las tablas a exportar (con los códigos del módulo) y sus opciones. Por ejemplo, si queremos exportar desarrollo longitudinal (código 0) de electrones y muones (códigos 16 y 17), con las opciones por defecto y *slanted*:

  `[[0, 15, ''], [0, 15, 'a'], [0, 16, ''], [0, 16, 'a']]`

- `extra`: Lista con todas las instrucciones que queramos incluir y no lo hayan sido hasta ahora. Por ejemplo, si queremos que los procesos fotonucleares y que sólo se registren datos del nivel de observación 30:

  `['PhotoNuclear Off', 'RecordObsLevels 30']`

- `save_path`: ruta del directorio donde queremos guardar el input file creado.

▪ `shell_script(job_name, input_file_dir, input_files, local_dir = '',`
`main = '/home2/', user = 'sergio.cabana/',`
`exe = ['aires/bin/ZHAireSRASPASS', 'SpecialPrimaries/RASPASSprimary'],`
`program = 'ZHAireSRASPASS', autorename = True)`

Crea un script con extensión `.sh` con instrucciones para ejecutar un input file concreto en un sistema de colas Sun Grid Engine. Muchas cosas (opciones de bash) están adaptadas para mi uso, pero puede cambiarse en el script del módulo fácilmente. Los argumentos son:

- `job_name`: Nombre para el trabajo que se envía al sistema de colas.
- `input_file_dir`: Ruta dentro de `/main/user/` en la máquina remota donde hemos subido los input files, y donde se guardarán los outputs.

- `input_files`: Lista con los nombres de los input files a ejecutar con este script.
- `local_dir`: directorio local para guardar el script `.sh` creado.
- `main`, `user`: directorios base para el usuario.
- `exe`: Lista con las rutas, dentro de `/main/user/`, con los ejecutables para correr simulaciones e incluir partículas especiales.
- `program`: El nombre del programa que corre los input files.
- `autorename`: Incluye loops para renombrar tablas exportadas con `Opt a` (i.e., cambia las extensiones acabadas en `b` por la extensión normal, y añade `_slan` o `_vert` en el nombre del archivo según el caso).

- `simulator(task_names, basics, trajects, sim_controls, exports, extras, jobID, RASPASS = False, upgoing = False, ZHAireS = False, ZHAireS_control = [], remote_main = '/home2/', user = 'sergio.cabana/', remote_dir = '', exe = ['aires/bin/ZHAireSRASPASS', 'SpecialPrimaries/RASPASSprimary', 'SpecialPrimaries/uprimary'], program = 'ZHAireSRASPASS', autorename = True local_savepath = '', eco = False, server = 'mastercr1.igfae.usc.es', node = 'nodo014', username = 'sergio.cabana')`

  Esta función crea los input files, los scripts `.sh`, los envía a la máquina remota y los pone a correr en el sistema de colas. Los inputs son:

  - `task_names`, `basics`, `sim_controls`, `exports`, `extras`: Listas de elementos como los que actuaban de inputs en `input_file`.
  - `jobID`: Un nombre para los trabajos que se mandan. Con el argumento `eco = True`, todos los inputs files se incluirán en un único script shell, y por lo tanto se ejecutarán en un único trabajo de nombre *jobID* en un solo nodo de la máquina. Si por el contrario `eco = False`, para cada input file generado se crea un archivo `.sh`, y tendremos un trabajo *jobID0, jobID1, jobID2, ...* para cada input file, corriendo cada uno en un nodo (y por tanto acabando en menos tiempo).
  - `RASPASS, upgoing, ZHAireS`: Los mismos bools que en `input_file`.
  - `ZHAireS_control`: Lista con los elementos de control de ZHAireS iguales a los que se usaban en `input_file`. Ahora tendremos uno para cada simulación a ejecutar.
  - `remote_main`, `remote_user`: Directorios base en la máquina remota.
  - `remote_dir`: Directorio dentro de `/remote_main/remote_user/` donde subir los input files, los scripts `.sh` y guardar los outputs.
  - `exe`: Lista con las rutas dentro de `/remote_main/remote_user/` a los ejecutables necesarios.
  - `program`: Nombre del programa que corre las simulaciones.
  - `autorename`: Renombrado de tablas con `Opt a`.
  - `local_savepath`: Ruta para guardar los archivos generados en la máquina local.

- `server`: Nombre del servidor remoto (conexión SSH).
- `node`: Nombre del nodo, dentro del servidor remoto, en que se ejecutan las simulaciones. La función está diseñada para un proceso como:

  $$\text{Archivos en local} \underbrace{\longrightarrow}_{\text{SSH, SFTP}} \text{Servidor remoto} \underbrace{\longrightarrow}_{\text{SSH}} \text{Nodo del servidor} \underbrace{\longrightarrow}_{\texttt{qsub}} \text{Ejecución y outputs}$$

  Este aspecto de la función, sin embargo, puede cambiarse fácilmente comentando algunas líneas (por ejemplo, para eliminar el salto a un nodo dentro del servidor).
- `username`: Usuario en la máquina remota.